Optimization of Multi-level Checkpoint Model with Uncertain Execution Scales

Sheng Di[12], Leonardo Bautista-Gomez[2], Franck Cappello[23]
[1]INRIA, France, [2]Argonne National Laboratory, USA,
[3]University of Illinois at Urbana-Champaign, USA
{sdi1, leobago, cappello}@anl.gov

***Abstract*—As for future extreme scale systems, there could be different types of failures striking exa-scale applications with different failure scales, from transient uncorrectable memory errors in processes to massive system outages. In this paper, we propose a multi-level checkpoint model by taking into account uncertain execution scales (different numbers of processes/cores). The contribution is three-fold. (1) We provide an in-depth analysis on why it is very tough to derive the optimal checkpoint intervals for different checkpoint levels and optimize the number of cores simultaneously. (2) We devise a novel method which can quickly obtain an optimized solution, which is the first successful attempt in the multi-level checkpoint model with uncertain scales. (3) We perform both large-scale real experiments and extreme-scale numerical simulation to validate the effectiveness of our design. Experiments confirm our optimized solution outperforms other state-of-the-art solutions by 4.3-88% on wall-clock length.**

I. Introduction

Extreme Scale HPC environment with a million cores or more will be used more and more commonly in solving scientific problems [1]. However, with such number of cores, HPC environments become more fragile and fault tolerance mechanisms need to be used to protect HPC executions.

To solve the fault-tolerance issue, many existing solutions [2] adopt periodic checkpoint/restart model (or checkpoint model for short) [3], [4], because of its easy-to-use nature and acceptable overall performance in practice [5], [6], [7]. For an MPI program, one just needs to periodically set checkpoints (i.e., store running processes' memories) via a *Parallel File System* (*PFS*), and roll back execution to the most recent checkpoint upon failures. Such a simple PFS-based checkpoint model is called single level, due to the use of only one storage technology for storing checkpoints.

For exascale applications, classic single level checkpoint model may result in huge performance degradation. On the one hand, exascale applications [8] are likely to be struck by different types of failures frequently, because of the number of cores used simultaneously. On the other hand, classic checkpoint/restart model always stores checkpoint files onto PFS, while exascale applications tend to use very large-size data sets during the execution, leading to huge checkpoint/restart overhead (up to 25%) [9] due to I/O bottleneck.

In comparison to single-level checkpoint/restart model, multi-level checkpoint/restart [10], [11], [12], [13] is a promising model to solve the above problem. Fault Tolerance Interface (FTI) [13], for example, is a toolkit which provides four-level checkpointing interfaces, including local-storage-device, partner-copy [13], [14], Reed-Solomon encoding (RS-encoding) [15], [16], and PFS. Different checkpoint levels handle different types of failure events. For example, partner-copy technology makes each checkpoint file have two copies stored in local storage device and another partner-node respectively. Thus, upon a failure event with multiple simultaneous hardware crashes[1], the whole execution can still be recovered via partner-copy as long as there are no adjacent/partner nodes crashed.

Since different levels correspond to various checkpoint overheads, such a multi-level checkpoint model creates a new avenue to refine the checkpoint strategies. For example, saving processes' runtime memories in the form of checkpoint files onto local storage devices in parallel would be much faster than onto PFS. The difference of checkpoint overheads between local storage devices and PFS will be larger in the near-future systems, especially due to the rapid development of Non-Volatile Dynamic RAM (NVDRAM) (such as DVDIMM [17]), which is now available with DDR3 DRAM [18] and to be integrated with DDR4 [19].

In our previous work [20], we proposed an efficient method to compute the optimal checkpoint intervals for different levels and also optimize the selection of levels for each HPC application. However, that solution optimizes the checkpoint intervals for fixed execution scales. Our recent experiments indicate that an application's performance with checkpoint model is not only determined by the checkpoint/recovery overheads and roll-back loss, but also related to the execution scale with consideration of checkpoint/recovery overhead and failure events. Hence, it is necessary to revisit the multi-level checkpoint model to optimize the execution scales based on multi-level checkpoint overheads and various failure events.

There are three key contributions in this work.

- We first reformulate and analyze the multi-level checkpoint model based on both checkpoint interval variables and the number of cores to be used in execution.

[1]*Simultaneous failures* means multiple nodes fail in a short period (a.k.a., correlated failure window), such as resource allocation period. In [22] and [23], the window lengths are set to 1 min. and 2 min. respectively.

Our problem formulation takes into account both linear speed-up applications and non-linear speed-up applications. The key challenge of the problem is two-fold: (1) the lower-level checkpoint overheads will impact the higher-level roll-back loss, so we have to combine all of checkpoint levels with different failure rates to optimize the entire performance for each HPC application; (2) the failure event probability likely depends on execution scales, that is, different numbers of cores in execution will definitely impact the failure rates on different checkpoint levels, which leads to a non-convex optimization problem. This means that it is extremely tough to find the optimal solution based on such a new multi-level checkpoint model. This contrasts Young's formula [3] or Daly's work [4], which is much easier to get approximate checkpoint intervals because of single checkpoint level and fixed execution scales.

- We propose a method that can quickly optimize the trade-off between speed-up and various overheads, with consideration of uncertain execution scales. That is, the new solution can optimize the checkpoint intervals for different levels and optimize the number of cores simultaneously. On the one hand, the larger number of processes/cores used, the shorter productive time we obtain in general. On the other hand, different numbers of processes to be launched in the execution will lead to different failure rates and checkpoint overheads to suffer. So, there would be an optimum scale for a specific application. Based on the multi-level checkpoint model, we comprehensively analyze how to optimize the scales with regard to both the application speedup and various overheads, which is the first attempt to solve this problem to the best of our knowledge.
- We evaluate our multi-level checkpoint model and some other related works, via an exascale simulation environment. This environment provides simulation results very close to practical situations, since we carefully emulate the execution of real MPI programs and also validate the correctness of its generated results using a real cluster environment with 1000+ cores. Our solution adopts both the optimized checkpoint intervals and optimized execution scales computed by our algorithm. The related works include (1) single-level PFS checkpoint model with Young's formula [3]; (2) multi-level checkpoint model with optimized checkpoint intervals yet without optimization of scales (i.e., our previous work [20]); (3) single-level PFS checkpoint model with both Young's formula and optimized number of processes, which is proposed by [21]. Experiments show that our approach outperforms other solutions by 4.3-88%.

The rest of the paper is organized as follows. In Section II, we formulate the multi-level periodic checkpoint model to minimize the entire wall-clock time for each HPC application with respect to execution scales and checkpoint intervals. In Section III, we derive the provably optimal solution for different types of HPC applications. We present our experimental results in Section IV, discuss the related works in Section V, and finally provide concluding remarks with a vision of future work in Section VI.

II. Problem Formulation

In this section, we formulate our research as an optimization problem based on multi-level checkpoint/restart model. We mainly focus on the periodic checkpoint model (a.k.a., equidistant checkpoint model), as it is a de-facto standard in the fault-tolerance research.

Here, we propose a generic multi-level checkpoint/restart model with L checkpoint levels. The checkpoint level 1 corresponds to transient/software failures. The remaining higher levels (2,3,$\cdots$,L) correspond to different cases of hardware failures. In FTI, for example, there are four checkpoint levels (local storage, partner-copy, RS-encoding, and PFS), which correspond to "software error with no hardware failure", "nonadjacent node failures", "a certain number of node failures with adjacent failure cases", and "the situations that lower levels cannot take over" respectively. Upon any type of hardware failure, the system will reallocate a new set of nodes/cores to replace the crashed nodes/cores, and the resource allocation is a constant period, denoted by A, which is far shorter than application execution time.

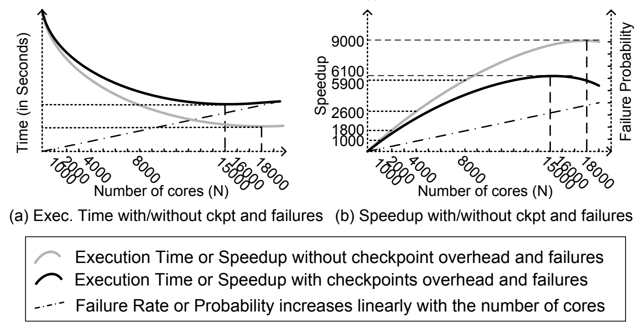
The checkpoint overhead and recovery overhead[1] are different from level to level and may also be different with various numbers of processes/cores used. Suppose we are given N processes running on N cores in parallel, the checkpoint overhead at the checkpoint level i is denoted by $C_i(N)$. In general, $C_1(N) \leq C_2(N) \leq \cdots \leq C_L(N)$. Similarly, the recovery overhead at level i is denoted by $R_i(N)$, where $R_1(N) \leq R_2(N) \leq \cdots \leq R_L(N)$ in general.

Considering the various checkpoint/recovery overheads with different numbers of cores, minimizing the wall-clock time for a specific HPC application is actually a tradeoff between its original speedup and increasing failure rates as the execution scales. In general, the speedup[2] of an HPC application increases more and more slowly with more cores (see grey curve in Figure 1). On the other hand, the failure rates may increase with the execution scales, shown as dash-dot lines in Figure 1. Figure 1 illustrates typical performance curves with and without regard to the checkpoint overheads and failure events. As shown by this figure, the optimal number of cores with checkpoint is lower than without checkpoint. In our paper, the application's real productive time with N cores is denoted as $f(T_e, N)$, where T_e refers to

[1]*Recovery overhead* means the time cost in restarting a failed application, so it is also known as restart overhead.

[2]*speedup*=$\frac{single\text{-}core\ length}{parallel\ execution\ time}$. For simplicity, we focus on the HPC applications whose speedups can be characterized as a continuous function instead of a piecewise function.

the single-core productive time (or execution length) by excluding any failure related costs such as checkpoint overhead and roll-back loss. We denote the speedup of the application running with N cores as $g(N)$. Hence, $f(T_e,N)$=$\frac{T_e}{g(N)}$. For example, if the parallel execution follows a linear speedup, then $g(N)=\kappa N$ and $f(T_e,N)\approx\frac{T_e}{\kappa N}$, where κ is a constant.



(a) Exec. Time with/without ckpt and failures (b) Speedup with/without ckpt and failures

Figure 1. Tradeoff between Execution Speedup vs. Checkpoint Overhead

All in all, the optimization of such a multi-level checkpoint model is non-trivial because of the following factors. (1) The entire wall-clock time of any application is synthetically determined by checkpoint/recovery overheads and roll-back losses at different levels. (2) The parallel execution may be struck by different types of failures, and the failure locations are randomly distributed. (3) The failure probabilities are different from level to level, and also related to the number of processes/cores to determine. In order to make the problem tractable, we introduce a key random variable called the number of failures (denoted by Y) during the application's execution.

Our objective is to minimize the expected wall-clock length $E(T_w)$ for each given application. $E(T_w)$ can be written as Formula (1), where L denotes the total number of checkpoint levels, x_i refers to the number of checkpoint intervals at level i, Γ_{ij} refers to the roll-back loss due to the jth failure occurring at level i in the execution, and $P_i(Y{=}K)$ denotes the probability of experiencing K failure events at checkpoint level i. We need to determine the optimal values of x_1,x_2,$\cdots$,x_L, and N, with minimized $E(T_w)$ regarding all possible overheads and roll-back losses.

$$E(T_w)=f(T_e,N)+\sum_{i=1}^{L}C_i(N)(x_i-1)$$
$$+\sum_{i=1}^{L}\left[\sum_{K=1}^{\infty}\left(P_i(Y{=}K)\sum_{j=1}^{K}\left(\Gamma_{ij}+A+R_i(N)\right)\right)\right] \quad (1)$$

Some key notations are summarized in Table I.

Such a multi-level checkpoint model is generic enough to be suitable for different scenarios. For example, the key difference between strong-scaling scenario and weak-scaling scenario is different speedup functions (either with consideration of scale grows or not) and checkpoint overhead/recovery functions. Our model is suitable for both of the two cases, because of generic definitions of the these functions in our model.

Table I
SUMMARY OF KEY NOTATIONS

Notation	Description
L	# of checkpoint levels each with various failure types
N	# of processes (or cores) of the focused application
T_e	the single-core productive time of the application
$f(T_e,N)$	the parallel execution time with N cores
$g(N)$	the speedup function of the parallel application
$C_i(N)$	checkpoint overhead at level i
$R_i(N)$	restart overhead at level i
A	resource allocation period (a constant)
x_i	# of checkpoint intervals of the application at level i
$P_i(Y{=}K)$	the probability of encountering K failures at level i
Γ_{ij}	roll-back loss in execution due to jth failure at level i

III. Optimization of Multi-level Checkpoint Model with Uncertain Execution Scales

In this section, we first theoretically analyze the huge challenges in solving the above multi-level checkpoint model, and then propose a method to optimize the solution.

A. Difficulty Analysis

One straight-forward idea is to leverage convex optimization theory [24]. Note that there are L+1 variables in the above problem, x_1, x_2, $\cdots$, x_L, N. It is necessary to prove the target function $E(T_w)$ is always convex with respect to any of the L+1 variables (i.e., $\forall$ i=1,2,$\cdots$,L, $\frac{\partial^2 E(T_w)}{\partial x_i^2}$>0 and $\frac{\partial^2 E(T_w)}{\partial N^2}$>0). Then, the optimal solution can be computed based on $\frac{\partial E(T_w)}{\partial x_i}$=0 and $\frac{\partial E(T_w)}{\partial N}$=0. Unfortunately, $E(T_w)$ is not always convex with respect to its variables.

In the following, we show that $E(T_w)$ is not convex even with respect to the **single-level** checkpoint model and linear-speedup application and, by generalization, to the **multi-level** checkpoint model with more complicated applications. That is, $g(\kappa)$=κN holds in the following analysis.

In the single-level checkpoint model with only PFS used to store checkpoint files, the checkpoint overhead and recovery overhead can be represented by Formula (2) and Formula (3) respectively. In the two formulas, ϵ_0, and η_0 refer to constant costs, α_0 and β_0 denotes two constant coefficients, and $H_c(N)$ and $H_r(N)$ denotes the increasing overhead rate with the execution scale. For instance, if the checkpoint/recovery overheads increase linearly then $H_c(N)$=$H_r(N)$=N holds; if they are constants, then $H_c(N)$=$H_r(N)$=0 holds.

$$C(N)=\epsilon_0+\alpha_0 H_c(N) \quad (2)$$

$$R(N)=\eta_0+\beta_0 H_r(N) \quad (3)$$

We will show that given $H_c(N)$=$H_r(N)$=N, the problem is already extremely tough to solve directly. We rewrite the problem formulation as Formula (4), where x is the number of checkpoint intervals and Γ_j refers to the roll-back loss upon the jth failure during the execution.

$$E(T_w)=\tfrac{T_e}{\kappa N}+(\epsilon_0+N\alpha_0)\cdot(x-1)$$
$$+\textstyle\sum_{K=1}^{\infty}\left(P(Y{=}K)\sum_{j=1}^{K}\left(\Gamma_j+A+\eta_0+N\beta_0\right)\right) \quad (4)$$

Since the failure events occur randomly during the task execution, the expected Γ_j (i.e., average rollback loss for each failure) can be approximated as $\frac{T_e/(\kappa N)}{2x}$, where $\frac{T_e/(\kappa N)}{x}$ refers to the maximum rollback length[1] upon a failure event. By further leveraging the definition of mathematical expectation (i.e., $\sum_{K=1}^{\infty}[K \cdot P(Y{=}K)] = E(Y)$), Formula (4) can be converted to Formula (5).

$$\begin{aligned} E(T_w) = \tfrac{T_e}{\kappa N} + (\epsilon_0 + N\alpha_0) \cdot (x-1) \\ +E(Y)(\tfrac{T_e/(\kappa N)}{2x} + \eta_0 + N\beta_0 + A) \end{aligned} \quad (5)$$

In general, the expected number of failures, $E(Y)$, is not a constant but related to the whole wall-clock length, and the expected wall-clock length is the variable $E(T_w)$ in the formula. If we denote the expected failure rate of the parallel application by $\lambda(N)$, then, $E(Y)$ can be approximated as $\lambda(N)E(T_w)$, which further leads to the following formula by eliminating $E(Y)$ from Formula (5).

$$E(T_w) = \frac{\frac{T_e}{\kappa N} + (\epsilon_0 + N\alpha_0)\cdot(x-1)}{1-\lambda(\frac{T_e}{2x\kappa N} + \eta_0 + N\beta_0 + A)} \quad (6)$$

As long as we can derive $\frac{\partial E^2(T_w)}{\partial x^2} \geq 0$ and $\frac{\partial E^2(T_w)}{\partial N^2} \geq 0$, we can compute the optimal solution based on $\frac{\partial E(T_w)}{\partial x}$=0 and $\frac{\partial E(T_w)}{\partial N}$=0. However, this is not a viable idea. On the one hand, it is hard to prove that the second-order derivatives $\frac{\partial E^2(T_w)}{\partial x^2}$ and $\frac{\partial E^2(T_w)}{\partial N^2}$ are always greater than 0 because of their extremely complicated representations. We find that they are actually lower than 0 in some situations. On the other hand, the first-order derivatives $\frac{\partial E(T_w)}{\partial x}$=0 and $\frac{\partial E(T_w)}{\partial N}$=0 also lead to very complicated high-degree equations (such as quartic equations), which is hard to solve [25].

B. Overview of Optimized Algorithm

We explore a very efficient algorithm to optimize the solution to the above problem. The basic idea is introducing an extra condition which assumes the expected number of failures (denoted by μ_i) at each specific level during the execution is only related to the execution scale (i.e., the number of processes/cores). That is, we can denote μ_i to be $\mu_i(N)$, and it is regardless of the application length. Then, we find that with such a condition, the target $E(T_w)$ can be simplified as a convex optimization problem with respect to all of its variables in general, so we can try to resolve a convex-optimization problem instead. The extra condition is removed later on, via a set of iterative steps: the algorithm alternatively derives optimal solution with the given numbers of failures and computes the new numbers of failures based on changed expected wall-clock length until convergence. The pseudo-code is shown in Algorithm 1.

In this algorithm, we first initialize the expected number of failure events for each checkpoint level i based on the

[1]For simplicity, we do not consider the situation a new failure event occurs during recovery period. In fact, based on our analysis in the previous work [20], failure-over-recovery situation occurs very rarely because recovery period is usually far shorter than productive time and failure interval.

Algorithm 1 OPTIMIZED ALGORITHM

Input: productive time T_e, estimated speedup function $g(N)$, checkpoint overheads, recovery overheads

```
1:  for (level $i$=1→$L$) do
2:      Compute $\mu_i$ based on $f(T_e, N)$ /*Initialize expected # of failures*/
3:  end for
4:  repeat
5:      Compute optimal solution $x_i^*$ ($i$=1,2,⋯,$L$) and $N^*$, based on
        convex optimization and $\mu_i$ ($i$=1,2,⋯,$L$).
6:      Compute expected $E(T_w)$ based on $x_i^*$ and $N^*$.
7:      for (level $i$=1→$L$) do
8:          $\mu_i'$←$\mu_i$. /*Save old $\mu_i$*/
9:          Recompute $\mu_i$ based on $E(T_w)$.
10:     end for
11: until $(\max(\mu_i' - \mu_i) \leq \delta, \forall i = 1, 2, \cdots, L)$
```

estimated productive time $f(T_e, N) = \frac{T_e}{g(N)}$ (line 1-3). Then, the algorithm goes into a loop that iteratively computes the optimal number of checkpoint intervals x_i^* for each level as well as the optimal number of cores N^*. In each iteration step, after computing the optimal solution with convex-optimization theory (line 5), the algorithm will adopt the solution to estimate the wall-clock length (line 6), based on which it can recompute more accurate expected numbers of failure events (line 7-10). This iteration loop will stop when the expected numbers of failure events converge within a specified error threshold δ. Obviously, the key step is line 5, which computes the optimal solution based on the expected numbers of failures on different levels which are regardless of the wall-clock length.

In the following text, we focus on how to compute the optimal solution with such an extra condition that the expected numbers of failures are regardless of wall-clock length. For improving the readability, we start with the situation with single-level checkpoint model, and then, we present the complete optimal solution for multi-level checkpoint model.

C. Optimization for Single-level Model

In this subsection, we investigate how to optimize checkpoint intervals and the execution scale simultaneously, for the linear-speedup applications and non-linear speedup applications respectively.

1) Optimization based on Linear Speedup: Here, we use linear-speedup application to show our basic idea. As for the linear-speedup applications, its productive time can be approximated as $f(T_w, N) {\approx} \frac{T_e}{\kappa N}$, where T_e denotes the single-core execution length and κ is a constant. In this case, we set $\mu(N)$=bN for simplicity, and will extend to more case functions in the multi-level model to be discussed later on. Similarly, we suppose the checkpoint/recovery overheads ($C(N)$ and $R(N)$) are constants for simplicity, and more cases will be discussed later on.

The representation of the target function $E(T_w)$ can be simplified as Formula (7).

$$E(T_w) = \tfrac{T_e}{\kappa N} + \epsilon_0(x-1) + bN(\tfrac{T_e/(\kappa N)}{2x} + \eta_0 + A) \quad (7)$$

Since $\frac{\partial E^2(T_w)}{\partial x^2}$=$\frac{bT_e}{\kappa x^3}$>0 and $\frac{\partial E^2(T_w)}{\partial N^2}$=$\frac{2T_e}{\kappa N^3}$>0, there must be an optimum point for the variables $\{x, N\}$ such that the expected wall-clock length $E(T_w)$ is minimized. We can compute the optimal values of x and N, as long as we solve both Formula (8) and Formula (9) simultaneously.

$$\frac{\partial E(T_w)}{\partial x} = \epsilon_0 + \frac{bT_e}{\kappa} \cdot \frac{-1}{2x^2} = 0 \tag{8}$$

$$\frac{\partial E(T_w)}{\partial N} = \frac{-T_e}{\kappa N^2} + b(\eta_0 + A) = 0 \tag{9}$$

Then, we can derive the following two simple formulas to directly compute the optimal number of checkpoint intervals x^* and the optimal scale N^*.

$$x^* = \sqrt{\frac{bT_e}{2\kappa\epsilon_0}} \tag{10}$$

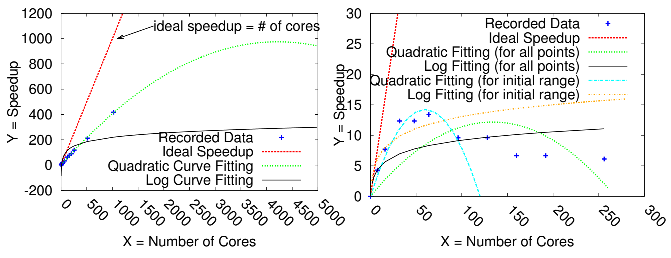
$$N^* = \sqrt{\frac{T_e}{\kappa b(\eta_0+A)}} \tag{11}$$

2) Optimization based on Non-linear Speedup: In our model, we propose a generic method to optimize the performance over the checkpoint model, which can fit different types/functions of speedup curves. In this work, we mainly focus on quadratic non-linear curves. However, our model can also be easily extended to more complicated speedup functions if needed, due to generic definition of speedup function $g(N)$. As an example, let us consider Figure 2 (a). The blue points shown in the figure are speedup values computed based on real experiments with up to 1024 cores on Argonne FUSION cluster [26]. The speedup of the *Heat Distribution* application[1] increases like a linear curve for the relatively small execution scale range, and follows a quadratic curve for the long range of execution scale. In fact, some applications' speedups may not follow quadratic curve in the whole execution range, as shown in Figure 2 (b). The speedup of Nek5000 eddy_uv application[2] increases quickly in the initial range while decreasing from the scale of 100 cores because of increasing communication cost. Since the optimal scale with regard to checkpoint model must be no bigger than the original optimal scale without checkpoint model, we just need to focus on the initial scale range through the point with maximum original speedup. Then, we can still use quadratic curve to fit the speedup very well, as shown in Figure 2 (b). Specifically, as shown in the figure, the quadratic curve generated based on the initial scale range (1-100 cores) fits the increasing speedup records best from among all different fitting curves.

The speedup function is defined as Formula (12). We denote the symmetrical axis of the quadratic curve to be located at $N^{(*)}$. Then, since the speedup curve must pass through the origin (0,0), we can derive the speedup curve as Formula (12), where N is the number of cores and κ is referred to as the curve's slope at (0,0). In practice, κ can be estimated by a simple test with a small or middle execution scale. In the above-mentioned MPI program Heat Distribution, for example, the speedup is 77 when using 160 cores, so κ can be approximated as $\frac{77}{160}\approx 0.48$, which is very close to the real value κ=0.46. In fact, the coefficients of Formula (12) can also be estimated via a set of laws such as Amdahl's Law [31], Gustafson-Barsis's Law [32], and Karp-Flatt metric [33], which are used to predict, respectively, the upper limit of the speedup, how long the application would take to run on a single core, and the performance of the application on larger execution scales.

[1]*Heat Distribution* computes the distribution of the heat for a room over time given a set of initial heat sources. The MPI program splits a particular space into several blocks and computes the heat distribution for each of them in parallel with communicated messages on the shared edges of the blocks. Such a parallel computation is commonly used in real scientific projects like Ocean Simulation [27], [28].

[2]This Nek5000 eddy_uv application monitors the error for a 2D solution to the Navier-Stokes equations [29].



(a) Heat Distribution (b) Nek5000 eddy_uv application

Figure 2. MPI Speedup and Curve Fitting

$$g(N) = -\frac{\kappa}{2N^{(*)}}N^2 + \kappa N \tag{12}$$

Similar to the derivation of Formula (7), the target function with respect to the non-linear speedup applications can be written as Formula (13). The only difference is that the execution speedup $g(N)$ in Formula (13) follows a non-linear curve as shown in Figure 2.

$$E(T_w) = \frac{T_e}{g(N)} + \epsilon_0(x-1) + bN\left(\frac{T_e/g(N)}{2x} + \eta_0 + A\right) \tag{13}$$

It is easy to verify that the target function Formula (13) is always convex, with respect to x and N respectively. Hence, there must be a unique value point for (x,N) with the minimum value of $E(T_w)$, and we can compute the optimal solution (x^*,N^*) by making $\frac{\partial E(T_w)}{\partial x}$=0 and $\frac{\partial E(T_w)}{\partial N}$=0. Accordingly, we derive the following equations:

$$\frac{\partial E(T_w)}{\partial x} = \epsilon_0 + \frac{bNT_e}{2g(N)}\left(\frac{-1}{x^2}\right) = 0 \tag{14}$$

$$\frac{\partial E(T_w)}{\partial N} = T_e\frac{b}{2x}\frac{1}{g(N)} - T_e\frac{(1+\frac{bN}{2x})g'(N)}{g^2(N)} + b(\eta_0 + A) = 0 \tag{15}$$

It is hard to directly solve the above two equations simultaneously, because the transformed equation after eliminating x using Equation (14) leads to a too complicated equation. Instead, we use a fixed-point iteration method to compute the solution to the above equation system. More specifically, we construct the following two iterative formulas based on Equation (14) and Equation (15), where (k) refers to the iteration index. Then, we can solve it by iteratively computing x and N until a convergence with an acceptable tiny error between $x^{(k)}$ and $x^{(k+1)}$.

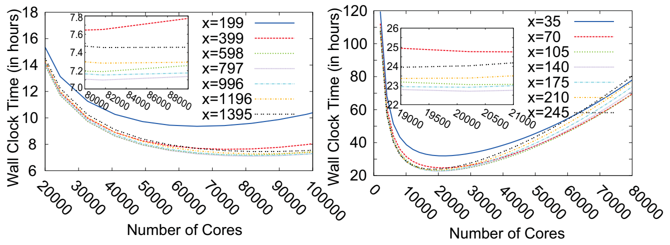
$$x^{(k+1)} = \sqrt{\frac{bN^{(k)}T_e}{2\epsilon_0 g(N^{(k)})}} \tag{16}$$

$$\frac{T_e b/x^{(k)}}{2g(N^{(k+1)})} - T_e \frac{(1+\frac{bN^{(k+1)}}{2x^{(k)}})g'(N^{(k+1)})}{g^2(N^{(k+1)})} + b(\eta_0+A) = 0 \quad (17)$$

In fact, it is still very challenging to solve Equation (17) directly, yet we can compute the approximate root $N^{(k+1)}$ for Equation (17) using Bisection method very efficiently. On the one hand, there must be an ideal number of cores (denoted by $N^{(*)}$) for any application because of inevitable synchronization/communication among processes. Since $\frac{\partial E^2(T_w)}{\partial N^2}$ must always be greater or equal to 0 in the range [0,$N^{(*)}$] and the optimal solution N^* must be no greater than $N^{(*)}$, there must be at most one root in [0,$N^{(*)}$] to satisfy $\frac{\partial E(T_w)}{\partial N}$=0. On the other hand, if there exists one root in [0,$N^{(*)}$], we can use Bisection method to approximate it, because the left-hand side of Formula (17) is a monotonically increasing function (note that $\frac{\partial E^2(T_w)}{\partial N^2}$>0). Since N^* must be an integer, the Bisection method can stop whenever the error of N between two adjacent steps (such as step k and step k+1) is smaller than 0.5, thus the convergence speed could be further improved (our simulation shows there are only about 10 iteration steps in general). If no root exists in [0,$N^{(*)}$], the optimal number of processes N^* must be equal to $N^{(*)}$. This situation occurs with very few failures or tiny checkpoint overhead on PFS.

Based on the Heat Distribution application speedup, we perform a numerical simulation to confirm the correctness of our optimal derivation. In the simulation, checkpoint overheads are based on our characterization of FTI over FUSION cluster. The coefficients of the speedup functions are computed using least squares method based on the real experimental data as shown in Figure 2. The error threshold is set to 10^{-6}, and x's initial value is set to 100,000. Our iterative method just needs 30-40 iterations to converge, which means a fairly high convergence speed.

Figure 3 presents different wall-clock times when running the Heat distribution application, comparing the optimal solution computed by our method to other comparative solutions. The workload to process is 4000 core-days and the original optimal execution scale $N^{(*)}$ is set to 100,000 cores. In addition, b=0.005, and κ=0.46. As for the two sub-figures, the checkpoint overheads are set to constant values ($C(N)$=$R(N)$=5 seconds) and linear-increasing values ($C(N)$=$R(N)$=5+0.005N) respectively. The optimal number of checkpoint intervals and the optimal number of processes/cores are 797 and 81746 respectively for the case with constant checkpoint cost, and their values are 140 and 20215 respectively for the case with linear-increasing checkpoint cost. Note that the original optimal execution scale without checkpoints and failure events is 100,000 cores for Heat Distribution, which means that the optimal execution scale changes prominently due to the impact of the failure rate that increases with the execution scale and checkpoint overheads. More experimental evaluation results can be found in Section IV.



(a) constant ckpt cost (b) linear-increasing ckpt cost

Figure 3. Confirming the Optimal Solution via Numerical Simulation

D. Optimization for Multi-level Model

Since there are multiple checkpoint levels involving the application's execution, we need to combine them in a target function, i.e., Formula (1). In comparison to the single-level checkpoint model, there are two critical differences for the derivation of the expected wall-clock length $E(T_w)$ in the multi-level checkpoint model.

- The roll-back loss in multi-level checkpoint model is different from that in traditional single-level checkpoint model. As the application is restarted based on a checkpoint at level i, the total roll-back loss at this level has to include all checkpoint overheads at lower levels in addition to the lost execution time. For example, when the application rolls back to a level-3 checkpoint, both of the level-1 checkpoint overheads and level-2 checkpoint overheads have to be counted in the roll-back loss. On the other hand, since all failures are unpredictable with random arrival locations and the checkpoints are taken periodically with equal distances, the expected value of Γ_{ij} can be represented as Formula (18), where $\frac{f(T_e,N)/(2x_i)}{f(T_e,N)/x_k}$ refers to the number of checkpoints at level k during the roll-back period.

$$\begin{aligned} E(\Gamma_{ij}) &= \frac{f(T_e,N)}{2x_i} + \frac{C_i(N)}{2} + \sum_{k=1}^{i-1}\left(\frac{\frac{f(T_e,N)}{2x_i}}{\frac{f(T_e,N)}{x_k}}C_k(N)\right) \\ &= \frac{f(T_e,N)}{2x_i} + \sum\nolimits_{k=1}^{i} \frac{C_k(N)x_k}{2x_i} \end{aligned} \quad (18)$$

- In the multi-level model, we need to investigate how to represent checkpoint overhead $C_i(N)$ and recovery overhead $R_i(N)$ on each level i. In general, if we save checkpoint files to local storage devices of running processes, the checkpoint overhead is rather a constant because these is no congestion; if we use PFS to save checkpoint files, the checkpoint overhead may increase with the execution scale because of more checkpoint files (thus more meta data) to be handled by PFS and inevitable congestion. Our characterization of the Heat distribution's checkpoint overheads on Argonne FUSION cluster (Table II) shows that the checkpoint overheads on level 1, 2, and 3 are very stable, while they increase prominently on PFS.

  Based on the characterization of checkpoint overheads shown in the above table, we define the checkpoint overhead $C_i(N)$ and recovery overhead $R_i(N)$ as Formula (19) and Formula (20) respec-

Table II
CHECKPOINT OVERHEAD OF FTI (IN SECONDS)

Exe. Scale	**Ckpt Cost (level 1−4)**			
128 cores	0.9	2.53	3.7	7
256 cores	0.67	2.54	4.1	8.1
384 cores	0.67	2.25	3.9	14.3
512 cores	0.99	3.05	4.12	21.3
1024 cores	1.1	2.56	3.61	25.15

tively, where i=1,2,$\cdots$,L refers to checkpoint level and N refers to the number of cores. ϵ_i, α_i, η_i, and β_i are four constant coefficients with respect to level i, and they can be derived by least squares fitting method. $H_c(N)$ and $H_r(N)$ are two baseline functions which always pass through (0,0). For example, $H_c(N)$=$H_r(N)$=0 means constant checkpoint/recovery overhead and $H_c(N)$=$H_r(N)$=N implies linear-increasing checkpoint/recovery overhead. All of the coefficients and baseline functions in Formula (19) and (20) are supposed to be determined based on characterization of checkpoint/recovery overheads with different execution scales. As for Table II, since the checkpoint overheads for the first three levels look like constants, $\alpha_1 = \alpha_2 = \alpha_3 = 0$ approximately holds.

$$C_i(N) = \epsilon_i + \alpha_i H_c(N) \tag{19}$$

$$R_i(N) = \eta_i + \beta_i H_r(N) \tag{20}$$

Now, let us derive the optimal solution (to determine the optimal checkpoint intervals for each level and optimize the execution scale) to our multi-level checkpoint model as follows. Combining Formula (1) and Formula (18), we derive Formula (21), where μ_i is the expected number of failures belonging to level i, as presented by Formula (22) in which $P_i(Y{=}K)$ denotes the probability of the number of failure events during the execution. Note that μ_i is actually a variate of the execution scale N because the failure probability may change with execution scales.

$$\begin{aligned} E(T_w) &= \tfrac{T_e}{g(N)} + \sum_{i=1}^{L} C_i(N)(x_i - 1) \\ &+ \sum_{i=1}^{L}\left[\mu_i\left(\tfrac{T_e/g(N)}{2x_i} + \sum_{k=1}^{i}\tfrac{C_k(N)\cdot x_k}{2x_i} + (A + R_i(N))\right)\right] \end{aligned} \tag{21}$$

$$\mu_i = E_i(Y) = \textstyle\sum_{K=1}^{\infty} K \cdot P_i(Y = K) \tag{22}$$

Similar to the single-level model presented in Section III-C, we can derive $\frac{\partial E^2(T_w)}{\partial x_i^2}$>0 and $\frac{\partial E^2(T_w)}{\partial N^2}$>0, so there must be a unique optimum point for $\{x_1,x_2,\cdots,x_L,N\}$ with minimized $E(T_w)$. Similar to the derivation of the optimal solution for single-level checkpoint model, we can get the optimal solution to such a multi-level checkpoint model as follows (based on first order necessary conditions):

$$\begin{aligned} \tfrac{\partial E(T_w)}{\partial x_i} &= C_i - \tfrac{\mu_i}{2x_i^2}\left(\tfrac{T_e}{g(N)} + \textstyle\sum_{j=1}^{i-1} C_j x_j\right) \\ &+ \tfrac{C_i}{2}\textstyle\sum_{j=i+1}^{L}\tfrac{\mu_j}{x_j} = 0 \end{aligned} \tag{23}$$

$$\begin{aligned} \tfrac{\partial E}{\partial N} &= \tfrac{T_e}{g^2(N)}\left((\sum_{i=1}^{L}\tfrac{\mu_i'}{2x_i})g(N) - (1 + \sum_{i=1}^{L}\tfrac{\mu_i}{2x_i})g'(N) \right) \\ &+ \sum_{i=1}^{L} C_i'(x_i - 1) \\ &+ \sum_{i=1}^{L}\left[\mu_i'(\sum_{k=1}^{i}\tfrac{C_k x_k}{2x_i} + A + R_i) + \mu_i(\sum_{k=1}^{i}\tfrac{C_k' x_k}{2x_i} + R_i')\right] = 0 \end{aligned} \tag{24}$$

Note that Formula (23) represents a set of equations, where i=1,2,$\cdots$,L. For simple representation, we use C_i and R_i to denote $C_i(N)$ and $R_i(N)$ respectively, which are computed by Formula (19) and Formula (20), and C_i' and R_i' denotes their derivatives respectively.

In principle, as long as we find the solution to the above system of L+1 simultaneous equations, we find the optimal solution to the multi-level checkpoint problem that expected numbers of failure events at different levels are assumed to be only related to execution scale N (i.e., μ_i=$\mu_i(N)$). However, it is extremely tough to directly solve it because of too high degrees in the equations and multiple variables (including x_1,$\cdots$,x_L and N). As mentioned previously, we can use fixed-point iteration to obtain the solution. Specifically, based on Formula (23) and Formula (24), we can construct their corresponding iterative functions and alternatively compute x_i and N based on Formula (23) and Formula (24), until each equation approximately holds with little error. The initial values of $\{x_1,x_2,\cdots,x_L\}$ (as shown in Formula (25)) can be obtained by Young's formula [3], in that it leads to the sub-optimal checkpoint interval result for a particular level i without taking into account the impact of checkpoint overheads at other levels. Based on our model, Young's formula can be approximately presented as Formula (25), where $\frac{T_e}{g(N)}$ denotes the expected productive time without checkpoint model and $C_i(N)$ refers to the checkpoint overhead at level i when running with N processes/cores.

$$x_i = \sqrt{\frac{\mu_i(N)T_e/g(N)}{2C_i(N)}} \tag{25}$$

As analyzed in Section III-B, the estimated wall-clock time may change with changed numbers of failures, which will change the expected numbers of failures at each level in turn. Hence, our complete solution iteratively computes the new wall-clock time based on changing expected numbers of failures for the application until convergence, as presented in Algorithm 1. The experimental results are presented in Section IV.

As for our key algorithm (Algorithm 1), one critical question is whether it can always converge eventually, or in what situations it cannot converge. In fact, this algorithm cannot converge only when failure rates are extremely high, which would not happen in reality. If failure rates (i.e., μ_i) are fairly high, the new value of $E(T_w)$ in each iteration would become much larger than its old value computed in last iteration, such that the new expected failure rates μ_i' may become unexpectedly larger in turn. In our evaluation (to be shown in next section), the failure rate is set up to 16+12+8+4=40 failures per day, which is already very high. Our Algorithm 1 can still converge quickly in this situation, which means that convergence issue is not a concern here.

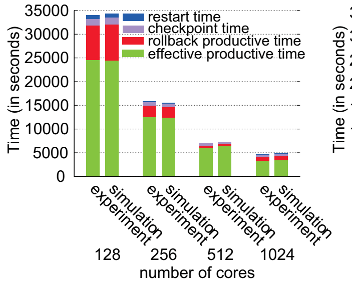
IV. Performance Evaluation

A. *Experimental Setting*

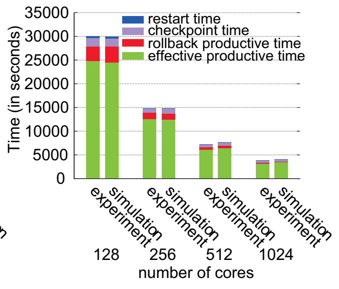
Since our fault-tolerance research is designed for exascale HPC applications, it is best to perform the evaluation with hundreds of thousands of real cores. However, there are at most 128 physical nodes (with totally 1024 cores) available in Argonne FUSION cluster [26] with limited resource usage quota, so we have to leverage exascale simulation to evaluate our multi-level checkpoint model. We perform practical experiments deployed with FTI and real MPI programs on Argonne FUSION cluster to confirm the high credibility and accuracy of our simulation environment.

The application used in our experiment is a well-known MPI program, called *Heat Distribution*, whose MPI communication methods (such as the ghost array between adjacent blocks) are commonly adopted in real HPC scientific projects like Parallel Ocean Simulation [27]. The key reason we adopted this application in our evaluation is that it is a real MPI program and it is also suitable to be executed with a large number of cores, as shown in Figure 2. It involves many MPI functions, including MPI_Bcast, MPI_Barrier, MPI_Recv, MPI_Send, MPI_Irecv, MPI_Isend, MPI_Waitall, MPI_Allreduce, etc. The checkpoint overhead and recovery overhead are both dependent upon two factors: (1) the program's memory sizes, which are determined by the problem size; and (2) the execution scale, i.e., the number of processes. Table II shows the checkpoint overhead of FTI with hundreds of cores on FUSION cluster [26].

The exascale simulation is described as follows. Each test is performed by running the MPI program for processing an amount of workload (T_e), which corresponds to the single-core productive time. Each test is driven by ticks (one tick is equal to one second in the simulation), simulating the whole procedure of running an MPI program with a non-linear speedup that follows Formula (12). We adopt the real checkpoint/restart overhead characterized on Argonne FUSION cluster [26], and we also take into account the possible jittering of checkpoint/restart overheads (with random error ratio up to 30%). The checkpoint overheads are shown in Table II, where the least-squares-fitting coefficients (ϵ_i,α_i) are computed as (0.866,0), (2.586,0), (3.886,0) and (5.5,0.0212) respectively. The simulator comprehensively takes into account possible overlapping of different operations/events under the checkpoint model, such as simultaneous occurrence of taking checkpoint/recovery operations and failure events. The whole simulation works very closely to real experiments, which is confirmed by Figure 4 (a) and (b) with various checkpoint intervals on the four different levels. With the same setting, the simulation results exhibit very similar to the results with real cluster environment, and the difference is less than 4%.



(a) ckpt intvl:1,3,5,7 min



(b) ckpt intvl:1,2,3,4 min

Figure 4. Confirming the Effectiveness of Simulator

We perform the evaluation for 6 different cases each with different failure rates at the four checkpoint levels. Given a certain number of cores simultaneously used as the baseline (denoted by N_b, where the subscript b refers to baseline), r_1-r_2-r_3-r_4 denotes there are r_1/r_2/r_3/r_4 failure events occurring per day at level 1/2/3/4 respectively. For example, 8-4-2-1 means there are 8 failure events occurring each day at level 1, 4 failure events occurring per day at level 2, etc. The real failure rates experienced actually increase with the number of cores proportionally, as compared to the baseline number of cores N_b which is always set to $N^{(*)}$=10^6. Each failure may occur randomly at any time in the whole wall-clock period, including productive time and checkpoint/recovery period. The failure intervals follow exponential distribution, because this is the behavior of the system for most of its lifetime [37]. All of results shown in the following text are mean values based on 100 runs for each case with random failure events.

There are four solutions to compare in our evaluation.

- *ML*(*opt-scale*): **multi-level** model with **optimized** checkpoint intervals and **optimized** scale N^*.
  - the proposed solution in this paper.
- *SL*(*opt-scale*): **single-level** model with **optimized** checkpoint intervals and **optimized** scale N^* simultaneously.
  - improved Young's formula based on [21].
- *ML*(*ori-scale*): **multi-level** model with **optimized** checkpoint intervals and **original** optimal scale $N^{(*)}$.
  - the previous work proposed in [20].
- *SL*(*ori-scale*): **single-level** model with **optimized** checkpoint intervals and **original** optimal scale $N^{(*)}$.
  - classic Young's formula [3].

There are two key indicators, which are *wall-clock time* and *efficiency* [30]. The wall-clock time includes productive time and all of overheads. We also present different time portions, including checkpoint overhead, restart overhead, and roll-back workload time. The efficiency is also called processor utilization, which is defined as the ratio of the wall-clock-time based speedup[1] to the number of processes/cores used.

B. Experimental Results

Figure 5 presents the four different portions of the running time, based on different checkpoint solutions, and wall-clock time is the sum of the four portions. The original optimal execution scale ($N^{(*)}$) is 1 million cores, and the workload amount to process (i.e., total single-core productive time) is 3 million core-days. We zoom in a particular area in Figure 5 for clear observation of the results. We also present the number of processes/cores used by the optimized-scale solutions (including ML(opt-scale) and SL(opt-scale)) in Table III. The other two solutions, ML(ori-scale) and SL(ori-scale), adopt the original optimal scale, i.e., 1 million cores, in the evaluation.

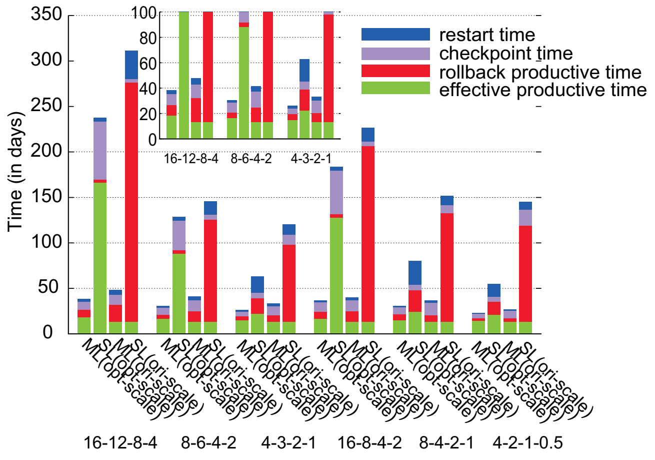


Figure 5. Time Analysis (T_e=3m core-days, $N^{(*)}$=1m cores)

Table III
OPTIMIZED EXECUTION SCALES IN MULTI-LEVEL MODEL AND SINGLE-LEVEL MODEL (T_e = 3 M CORE-DAYS, $N^{(*)}$ = 1 M CORES))

Solution	16-12-8-4	8-6-4-2	4-3-2-1	16-8-4-2	8-4-2-1	4-2-1-0.5
ML(opt-scale)	472k	564k	658k	563k	657k	734k
SL(opt-scale)	41k	78.6k	36.7k	53.6k	325k	399k

Based on Figure 5 and Table III, we have three important findings. Firstly, we observe that the total wall-clock time decreases with decreasing number of failure events (e.g., from 16-8-4-2 to 4-2-1-0.5), which is reasonable because of reduced total checkpoint/restart overheads and roll-back losses with less failures. Secondly, it is observed that our solution ML(opt-scale) always outperforms other approaches significantly. The wall-clock times of our multi-level model with optimized execution scale (N^*) can be reduced by 58-84%, 7-26%, and 79-88% than the other three solutions respectively. Lastly, we analyze as follows the key reasons why our solution significantly outperforms other approaches, based on the different time portions. As for SL(ori-scale), since it only adopts PFS to store checkpoint files and uses all of the 1 million processes/cores to perform the execution, the checkpoint/recovery overheads would be higher than those in multi-level model. On the other hand, the failure rates would also be fairly high due to too many cores being used simultaneously, which will cause an extremely large amount of roll-back loss. In comparison to SL(ori-scale), SL(opt-scale) optimizes the execution scales to avoid too many failure events during the execution, so it suffers from very low roll-back loss as shown in the figure, while its productive time is inevitably fairly long because of significantly reduced execution scales (e.g., only 41k cores are used for 16-12-8-4 use case). Comparing our new solution ML(opt-scale) to our previous work ML(ori-scale) [20], we can observe that (1) the productive time is always extended more or less because of the smaller execution scales used in the execution; (2) the rest three portions of times are all significantly reduced in our new solution, finally leading to the significant performance improvements.

In fact, we observe that our solution ML(opt-scale) may not always lead to huge performance gains than our previous approach ML(ori-scale) which uses all of available cores. For example, when running the application with 10 million core-days, the wall-clock length under ML(opt-scale) is less than that of SL(ori-scale) by 4.3-42.3%. The details can be found in Figure 6. The relatively degraded performance gains (as compared to the test with T_e=3m core-days) is mainly due to significantly longer productive time, which takes relatively more portions in the total wall-clock length.

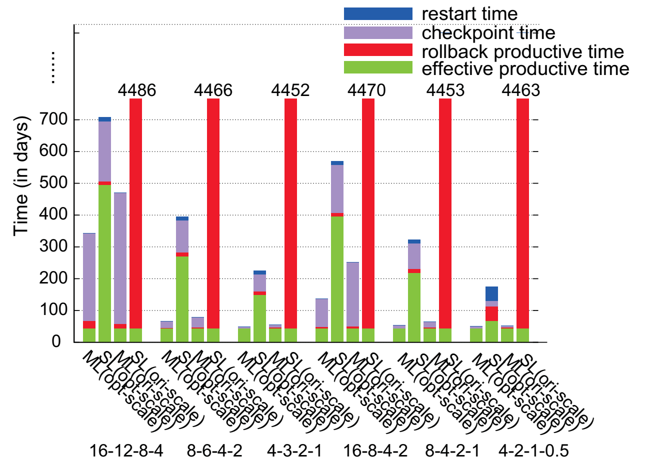
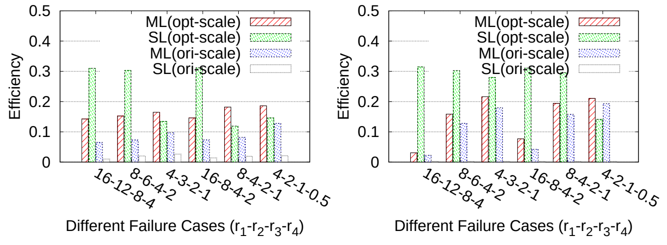


Figure 6. Time Analysis (T_e=10m core-days, $N^{(*)}$=1m cores)

Figure 7 shows the efficiencies of the above two tests with different workloads to process. It is observed that the single-level model with the optimization of execution scales (i.e., SL(opt-scale)) leads to highest efficiency, which is especially because of too few cores used in execution, as shown in Table III. This solution could save a large amount of available

[1]The speedup here refers to the ratio of the failure-free single-core productive time to the wall-clock time (including parallel productive time and all of overheads). Note that it is different from the original speedup, which is without consideration of failures and checkpoint/recovery overheads.

resources and energies like electricity power, however, it is definitely not preferred by users because of over-long wall-clock length induced (as shown in Figure 5 and Figure 6). In comparison to SL(opt-scale), our ML(opt-scale) solution results in much shorter wall-clock time and also keeps a relatively higher efficiency than other solutions, which can satisfy both users and system managers. Specifically, Table III shows that our solution just uses 40-79% cores from all 1 million cores, which significantly saves the resources and improves the system availability.



(a) T_e=3m core-days (b) T_e=10m core-days

Figure 7. Efficiencies of the Four Solutions with Different Cases

We also evaluate the checkpoint model, provided that the PFS checkpoint overhead is a constant as the storage scale increases, which occurs when using some special file system like Blue Water File System [38]. Although PFS can perform parallel I/O to improve its read/write rate, the total checkpoint overhead on PFS will still be much larger than that on local storage devices, because of much larger total checkpoint file size handled by PFS. Accordingly, we suppose the problem size is very huge such that checkpoint overheads on the four checkpoint levels are relatively large: 50, 100, 200, 2000 seconds respectively. Table IV presents the evaluation results based on the setting with T_e = 2 million core-days and $N^{(*)}$ = 1 million cores. Based on this table, it is observed that ML(opt-scale) always leads to the highest performance among all solutions. In particular, its wall-clock time (WCT) is shorter than that of ML(ori-scale) by 3.6-6.5%, and its efficiency is higher than that of ML(ori-scale) by 12.9-22.1%. The execution scales optimized by ML(opt-scale) are 860k-940k cores, which improves the system availability by 6-16% in comparison to using up all of available resources.

Table IV
Evaluation of Multi-level Ckpt Model with Constant Ckpt Cost on PFS: Wall-Clock Time (WCT) and Efficiency

Sol. WCT	**16-12-8-4**		**8-6-4-2**		**4-3-2-1**	
	WCT	**Eff**	**WCT**	**Eff**	**WCT**	**Eff**
ML(opt-scale)	14.6	0.158	12.8	0.173	11.1	0.193
SL(opt-scale)	37.3	0.092	23.2	0.123	17.2	0.146
ML(ori-scale)	15.4	0.13	13.4	0.15	11.7	0.171
SL(ori-scale)	890	0.002	892	0.002	890	0.002
ML(opt-scale)	13.1	0.171	11.7	0.186	10.6	0.2
SL(opt-scale)	30.6	0.10	20.4	0.133	16	0.153
ML(ori-scale)	14.2	0.14	12.2	0.164	11.4	0.176
SL(ori-scale)	893	0.002	890	0.002	896	0.002

Finally, we check the number of iterations used to converge the estimated failure rate with changing wall-clock length in Algorithm 1. The error threshold is set to 10^{-12}. For the above three evaluation cases, our algorithm just costs 8 iterations, 7 iterations, and 15 iterations respectively, which confirms fairly fast convergence speed.

V. Related Work

Optimization of exascale parallel computing performance based on checkpoint/restart model is a fundamental and quite challenging issue, which has been studied extensively in recent years [5], [6], [7], [14], especially with ever-increasing demand on exascale execution environment [1]. Some basic ideas (e.g., diskless checkpoint [11], [14]) are trying to reduce the checkpoint overheads as much as possible in the exascale environment, such that the checkpoint/restart model can still be kept effective with respect to the entire performance. Many other researchers proposed different models to take over the exascale resilience issue.

Multi-level checkpoint/restart model with different levels of checkpoint overheads has been proposed to provide an elastic response to tolerate different types of failures. SCR [12] is the first library which can be leveraged to checkpoint and recover HPC applications based on four storage levels (RAM, Flash, disk, and PFS). It also explores a Markov model to optimize the checkpoint strategies. However, they did not take into account the impact of the number of processes/cores to the execution performance, thus the proposed Markov model is not optimized with respect to the execution scales. In comparison to the SCR, FTI [13] is another outstanding library which also supports RS-encoding technology [15], [16]. It allows to recover the application from lightweight checkpoint files in case of multiple simultaneous hardware failures. Yet, FTI itself does not help optimize the checkpoint intervals. In our previous work [20], we proposed a multi-level checkpoint/restart model based on FTI, in which the checkpoint intervals are optimized for different checkpoint levels. It took into consideration many facets like various checkpoint/recovery overheads and failure rates on different levels, however, the application execution scale (i.e., the number of processes) was not considered in the optimization problem. In comparison to the above existing work, we propose a more comprehensive multi-level checkpoint model in this paper, which can optimize the checkpoint intervals and the execution scales simultaneously.

In addition, some other papers also discussed how to simultaneously optimize execution scales and checkpoint intervals for HPC applications. A typical method was proposed by Jin et al. [21]. There are many key differences between their work and our work. (1) Their optimization research is based on single-level checkpont/restart model, which is much simpler than multi-level checkpoint model tackled by our work. (2) Their work is not based on a real checkpoint toolkit and MPI applications, while ours stems from real-world characterization over multi-level checkpoint toolkits. (3) Their optimal checkpoint interval is derived by assuming failure's exponential distribution and using first-order approximation, while ours is without such assumption and approximation. (4) Our model is more concise: only with about 10 necessary notations (as shown in Table I), in comparison to the 30+ notations defined in their model. (5) They also derived the target function $E(T_w)$, while they did not prove that it is a convex function with respect to the variables before using Newton's method to approximate the optimal solution. In that situation, the converged result may not be globally optimized, and even worse, the algorithm may not converge given unappropriate initial values. In contrast, we comprehensively analyzed the multi-level checkpoint model with uncertain execution scales, and conduct a set of exascale simulations to validate its performance using real-world MPI programs and real checkpoint/recovery overheads.

VI. Conclusion and Future Work

In this paper, we improved our multi-level checkpoint/restart model by optimizing the checkpoint intervals for different checkpoint levels and the number of processes simultaneously. Such a new problem is extremely tough to solve based on our analysis, not only because of synthetic performance related to different types of failures but also due to the varied failure probability with uncertain number of processes/cores to determine. However, we still successfully devise an algorithm that can obtain the optimal solution efficiently. Some key findings are listed below.

- Our algorithm just requires 7-15 iterations to converge, which means a pretty fast computation speed.
- The optimized execution scale is smaller than the original optimal scale by 40-95%, which can significantly improve system availability.
- Our solution outperforms other solutions by 4.3-88% on wall-clock length, and improves efficiency by 12.9+%.

In the future, we will explore how to optimize the checkpoint strategies for more complicated non-linear applications.

Acknowledgments

This work was supported by PRACE First implementation Phase (PRACE-1IP) as the AMFT prototype under contract RI-261557 and by GENCI, also in part by the Advanced Scientific Computing Research Program, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, and by the ANR RESCUE, ANR G8 ECS projects and the INRIA-Illinois Joint Laboratory for Petascale Computing.

References

[1] A. Feinberg. (2013). An 83,000-Processor Supercomputer Can Only Match 1% of Your Brain. [online]. Available: http://gizmodo.com/an-83-000-processor-supercomputer-only-matched-one-perc-1045026757.

[2] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward Exascale Resilience," *International Journal of High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 374–388, 2009.

[3] J.W. Young, "A first order approximation to the optimum checkpoint interval," *Communications ACM*, vol. 17, no. 9, pp. 530–531, 1974.

[4] J.T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computater Systems*, vol. 22, no. 3, pp. 303–312, 2006.

[5] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, Y. and F. Vivien, "Checkpointing strategies for parallel jobs," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, 2011, pp. 1–11.

[6] K. Pattabiraman, C. Vick, and A. Wood, "Modeling Coordinated Checkpointing for Large-Scale Supercomputers," in *Proc. International Conference on Dependable Systems and Networks (DSN'05)*, 2005, pp. 812–821.

[7] K. Ferreira, "Keeping Checkpoint/Restart Viable for Exascale Systems," Ph.D Thesis, Computer Science, University of New Mexico, 2011.

[8] E. Vivek Sarkar, et al., "Exascale Software Study: Software Challenges in Exascale Systems, " technical report, 2009.

[9] B. Schroeder and G. Gibson, "Understanding Failure in Petascale Computers," *Journal of Physics Conference Series: SciDAC*, vol. 78, pp. 11–22, June 2007.

[10] L. Bautista-Gomez, A. Nukada, N. Maruyama, and F. Cappello, and B. Matsuoka, "Low-overhead diskless checkpoint for hybrid computing systems,", in *Proc. International Conference on High Performance Computing (HiPC'10)*, 2010, pp. 1-10.

[11] L. Bautista-Gomez, N. Maruyama, F. Cappello, and S. Matsuoka, "Distributed Diskless Checkpoint for Large Scale Systems," in *Proc. 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'10)*, 2010, pp.63-72.

[12] A. Moody, G. Bronevetsky, K. Mohror, and B.R. Supinski, "Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System," in *Proc. of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, 2010, pp. 1-11.

[13] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, and N. Maruyama, S. Matsuoka, "FTI: high performance fault tolerance interface for hybrid systems," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, 2011, pp. 32:1–32:32.

[14] G. Zheng, X. Ni, and L.V. Kale, "A scalable double in-memory checkpoint and restart scheme towards exascale," in *EEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W12)*, 2012, pp 1-6.

[15] I.S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, pp.300–304, 1960.

[16] J.S. Plank, S. Simmerman and C.D. Schuman, "Jerasure: A Library in C/C++ Facilitating Erasure Coding for Storage Applications - Version 1.2," University of Tennessee, Technical Report, Aug. 2008, CS-08-627.

[17] DVDIMM technology page. [online]. Available: http://en.wikipedia.org/wiki/NVDIMM.

[18] Marvell DragonFly NVRAM: Second-Generation, High-Performance Non-Volatile DRAM Write Cache. [online]. Available: http://www.marvell.com/storage/dragonfly/assets/Marvell_DragonFly_NVRAM-02_product_brief.pdf

[19] G. Findley, C. Johnson, R. Sethi, M. Howard, and S.S. Miguel. Panel: DDR4 Memory Ecosystem. (2013). [online]. Available: https://intel.activeevents.com/sf13/connect/fileDownload/ session/D0E05F6EFB3E53CD156AAE2DB378BD83/SF13_SPCP001_100.pdf.

[20] S. Di, M.S. Bouguerra, L.B. Gomez, F. Cappello, "Optimization of Multi-level Checkpoint Model for Large-scale HPC Applications," in *Proc. International Parallel and Distributed Processing Symposium (IPDPS 2014)*, 2014.

[21] H. jin, Y. Chen, X. Sun, "Optimizing HPC Fault-Tolerant Environment: An Analytical Approach," in *Proc. International Conference on Parallel Processing (ICPP'10)*, 2010, pp. 525–534.

[22] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello, "Modeling and tolerating heterogeneous failures in large parallel systems," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis(SC'11)*, 2011, pp. 45:1-45:12.

[23] D. Ford, F. Labelle, F.I. Popovici, M. Stokely, Murray, V.A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. 9th USENIX conference on Operating systems design and implementation (OSDI'10)*, 2010.

[24] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2009.

[25] Solving Quatic Equation. [online]. Available: http://en.wikipedia.org/wiki/Quartic_function

[26] FUSION Cluster. [online]. Available: http://www.lcrc.anl.gov/

[27] R. Smith, et al. "The Parallel Ocean Program (POP) Reference Manual: Ocean Component of the Community Climate System Model (CCSM)," technical report, Los Alamos National Laboratory (LAUR-10-01853), 2010.

[28] Community Earth System Model (CESM). [online]. Available: http://www2.cesm.ucar.edu

[29] O. Walsh, "Eddy Solutions of the Navier-Stokes Equations, " *Navier-Stokes Equations II - Theory and Numerical Methods*, vol.1530, pp. 306–309, 1992.

[30] M. Quinn, *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Science/Engineering/Math. ISBN 0072822562, 2005.

[31] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities, " in *AFIPS Conference Proceedings*, vol.30, 1967, pp. 483–485.

[32] J Gustafson, "Reevaluating Amdahl's Law, ", John L. Gustafson, *Communication of the ACM*, vol.31, no.5, pp. 532-533, 1988.

[33] A. Karp, and H. Flatt, "Measuring Parallel Processor Performance," *Communication of the ACM*, vol.33, no.5, pp. 539C543, 1990.

[34] P.F. Fischer, J.W. Lottes and S.G. Kerkemeier. nek5000 Web page. " [online]. Available: http://nek5000.mcs.anl.gov, 2008.

[35] Jacobi method in solving linear equestions. [online]. Available: http://en.wikipedia.org/wiki/Jacobi_method

[36] S. Di, Y. Robert, F. Vivien, D. Kondo, C-L. Wang, and F. Cappello, "Optimization of cloud task processing with checkpoint-restart mechanism," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*, 2013, pp. 64:1–64:11.

[37] D.L. Snyder and M.I. Miller, *Random Point Processes in Time and Space*. Springer-Verlag. ISBN 0-387-97577-2, 1991.

[38] K. Chadalavada and R. Sisneros, "Analysis of the Blue Waters File System Architecture for Application I/O Performance, " Cray User Group Meeting (CUG 2013), Napa, CA, May, 2013.